A Task Scheduling Algorithm for Multi-Core-Cluster Systems

Xiaozhong Geng [1,2]
[1]Department of Computer Science & Technology Jilin University, Changchun, China
[2]School of Electrical & Information Technology Changchun Institute of Technology, Changchun, China
gengxiaozhongcn@sina.com

Gaochao Xu[1,*], Xiaodong Fu[1], Yuan Zhang[2]
[1]Department of Computer Science & Technology Jilin University, Changchun, China
[2]Information center of State Food and Drug Administration, Beijing, China
* Corresponding author
xugc@jlu.edu.cn, 178143370@qq.com, zhangyuan2u@gmail.com

***Abstract*—The quantity of cores on one chip increases rapidly with the development of multi-core technology, which has led to more complex structure of cluster system and greatly increasing number of tasks. In order to schedule tasks in multi-core-cluster systems efficiently, a task schedule model based on the directed acyclic graph(DAG) is built, and then a algorithm based on task duplication is proposed. The algorithm is composed of two steps of operations, in which the processes are assigned to processor nodes in the first step and the threads in processes are assigned to core nodes in the second step respectively. The time complexity of this algorithm is less than similar algorithms. For the algorithm, minimization scheduling length is the primary objective, and keeping load balancing between processing nodes is secondary objectives. It can be seen through comparison with correlative work that the algorithm has advantages in scheduling length; furthermore, while the ratio of total communication cost and total computation cost in the task schedule model becomes larger, the advantage of this algorithm is more obvious.**

***Index Terms*—task duplication, task scheduling, multi-core processor, scheduling length, DAG, multi-core-cluster systems**

I. INTRODUCTION

Multi-cores architectures are becoming a mainstream in both server processors and desktop processors. Over the next decade, we expect to see processors with tens and even hundreds of cores on a chip[1].

With the architecture of high-performance general microprocessor into the multi-core era, multi-core processors have been increasingly used in cluster systems, Thus multi-core-cluster systems appears. Each processing node in multi-core-cluster systems is consisted of a multi-core processor; and the interconnection between nodes is through some network connection. Multi-core-cluster systems improve overall system performance by improving the performance of nodes, so that it has with good flexibility and scalability, which makes more computing cores with relatively higher cost performance ratio come true. Figure 1 shows a structure of a multi-core cluster system.

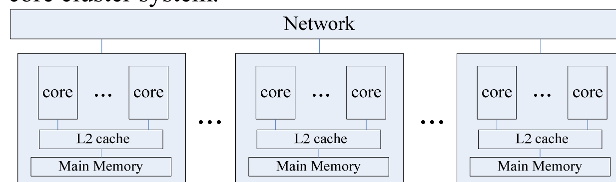

Figure 1. Structure of a multi-core cluster system.

Multi-core clusters include a two-level storage mechanism: Shared memory inside one node and distributed storage between nodes; and they include two-layer communication structure: communication between cores in a multi-core processor(CMP) and communication between processors. The relationship between these cores in a CMP is tightly coupled, and these cores are often interconnected by shared-cache or the high-speed data channel; but the relationship between processor nodes is loosely coupled, and these nodes are often interconnected by interconnection network, therefore, communication overhead between processor nodes is much larger than that between cores.

In the parallel programming model, the programming model based on message passing (such as MPI, PVM) is still a mainstream[2]. Parallel program based on the model contains multiple task(processes), and communication between tasks is by message passing. Under the environment of multi-core-cluster systems, in order to utilize computing resources more efficiently, each process may consist of multiple concurrent threads, which can run on different processor cores, and the communication between threads is via shared memory. Because address space of threads which belong to the different process is independent, communication between these threads is only through message passing.

Multi-core cluster is similar to traditional multi-processor cluster in structure, so we can regard each processor node (a CMP) as a cluster at the chip-level implementation. The mainly difference between them is that: Task scheduling of traditional clusters is that the

© 2012 ACADEMY PUBLISHER
doi:10.4304/jcp.7.11.2797-2804

parallel tasks will be allocated to each processor, and only considers the communication between processor; but task scheduling of multi-core-cluster systems is that the parallel tasks will be allocated to each core eventually, and considers the communication overhead both between processors and between cores in the allocation process. Therefore, we can improve and develop task allocation and scheduling algorithm for traditional clusters to adapt multi-core-cluster systems.

This paper discusses the task scheduling problem in multi-core-cluster systems, builds a task scheduling model, and then proposes a task scheduling algorithm based on task duplication. Comparison test shows that the algorithm can obtain near-optimal solutions in reasonable time, and behave even better in scalability than genetic algorithms. While the ratio of total communication cost and total computation cost in the task schedule model becomes larger, the advantage of this algorithm is more obvious.

The organization of the paper is as described below. Section 2 summarizes the related research works. Section 3 gives the task scheduling model on the basis of DAG. Section 4 proposes a task scheduling algorithm which consists of two steps of operations. Section 5 calculates time complexity of the proposed algorithm. Section 6 presents experimental results by comparing the algorithm with genetic algorithm. The last section concludes the paper with explanation of advantages and issues of the algorithm.

II. RELATED WORKS

The problem of task scheduling has been studied for many years. Most of scheduling algorithms are based on a task precedence graph (TPG) or a task interaction graph (TIG). TPG has been described by directed acyclic graph (DAG), in which the node represents a task and the edge represents task execution sequence and communication overhead. The DAG is usually used for static task scheduling model.

A task system is composed of the subtasks which have partial order relations and can be run in parallel or serial. The goal of task scheduling for traditional clusters is that subtasks be reasonably assigned to each processor according to some strategies and executed in parallel or serial, thus the communication overhead and latency between the parallel execution tasks is reduced, and thus task system execution time is cut short.

Task scheduling algorithm can be divided into two categories, static scheduling and dynamic scheduling. Compared with dynamic scheduling, static scheduling algorithm is simpler and with lower overhead, etc. There are two kinds of basic static scheduling algorithm: one is based on random search and the other is based on heuristics[3]. The algorithm based on the random search includes genetic algorithm[4], annealing algorithm[5], local search technology[6], etc.

Since it has been proved that the scheduling problem of multiprocessor is NP-complete, many researchers have proposed scheduling algorithms based on heuristics[7]. The algorithm based on heuristic includes table scheduling algorithm, cluster algorithm, task duplication algorithm, etc.

The basic idea of the task replication algorithm is to change the task communication from between different processing nodes to the same node by running the copy of a task which sends messages on multiple processing nodes which receive messages. The algorithm reduces the communication time between processing nodes by increasing local processing time of a node. When reasonable duplication strategies are adopted, task duplication algorithm can obtain better scheduling performance than other algorithms[8]. In this paper, an algorithm based on task duplication is adopted.

There are multiple typical task duplication algorithm such as TDS[9], OSA[10], PPA[11], CPFD [12], etc. The main idea of TDS algorithm is to allocate one friendly parent task and many subtasks to the same processing node, so that subtasks can be start at the earliest start time, which will shorten the schedule length. OSA algorithm is the evolved edition of TDS, and which allows many parent tasks and subtasks allocated to the same processing node, thus the earliest start time ever of subtasks is obtained, and then schedule length can be shorten. PPA algorithm has improved the OSA algorithm: using the same scheduling policy, it also optimizes the number of processing nodes. CPFD algorithm makes use of exploratory strategy, before the current task is scheduled, the algorithm recursively searches its VIP(Very Important Parent) [13] task out and copy it to the current processing node, which makes the current task to get the smallest and earliest start time, and thus schedule length can be shorten.

TDS algorithm does not allow multiple parent tasks and subtasks to be assigned to the same processing node; therefore, it is difficult for subtasks to get a good earliest start time. OSA, PPA, and CPFD algorithm can assign multiple parent tasks and subtasks to the same processing node, so the current task can get minimum start time, but that also limits the task scheduling of its descendant or ancestral task, and restricts optimization of scheduling length.

So far, the research on task scheduling algorithm for multi-core-cluster systems hasn't been developed enough. Task scheduling algorithm specializing in this system is relatively few.

[14] mixed ant colony algorithm (ACO) and genetic algorithm (GA) to execute task allocation and scheduling , but their system model is a multi-core system of global and local storage.

[15] proposed a task allocation algorithm based on iteration, which takes full advantage of the characteristics of multi-core systems, and reduces communication overhead during tasks execution process; however, the algorithm fails to consider the precedence relation between tasks.

[16] proposed a task scheduling algorithm for multi-core processor, which considered parallel tasks execution time and communication overhead, etc. This scheduling algorithm is efficient and can adapt to quantity alteration

© 2012 ACADEMY PUBLISHER

of processing cores, but it only apply to single processor system.

There is also some research for task allocation of multi-core DSP processor and network processors[17], but that does not apply to multi-core-Cluster systems because that only takes into account single processor system and fine-grained parallel.

III. Task Scheduling Model

Suppose the multi-core-cluster system is composed of N_{node} processor nodes $D_0, D_1, \ldots D_{N_{node}-1}$, and each processor is composed of N_{core} processing cores $C_0, C_1, \ldots C_{N_{core}-1}$; the parallel application to be allocated is composed of N_{proc} processes $P_0, P_1, \ldots P_{N_{proc}-1}$, and process P_i is composed of M_i threads $T_0, T_1, \ldots T_{M_i-1}$.

Then the total number of threads: $N_{thread} = \sum_{i=1}^{Nproc-1} M_i$.

A parallel program usually represented by a DAG, which is also called a task graph. A DAG consists of a tuple G = (T, E, R, W), where:

1) T is the set of vertices $\{T_i\}$, T_i is the corresponding task number to a vertex.

2)E is the set of directed edges $\{E_{ij}\}$. E_{ij} stands for the edge from task T_i to T_j, $E_{ij} \in E$ shows that, T_j can't be executed until T_i has been completed. Here T_i is called a predecessor of T_j while T_j is a successor of T_i .

3)R is the set of execution time of task vertex $\{R(T_i)\}$. $R(T_i)$ is the additional information of a vertex, which shows the execution time of task T_i.

4) W is the set of communication overhead between two tasks, which is expressed by $\{W(E_{ij})\}$. $W(E_{ij})$ is the additional information of an edge, which shows communication overhead between task T_i and task T_j.

5) $pred(T_i)=\{T_j|E_{ji} \in E\}$; $|pred(T_i)|$ is the number of predecessor tasks of T_i;

$succ(T_i)=\{T_j|E_{ij} \in E\}$; $|succ(T_i)|$ is the number of successor tasks of T_i;

If $|pred(T_i)| = 0$, T_i is start task, which is denoted by T_s;

If $|succ(T_i)| = 0$, T_i is end task, which is denoted by T_e;

If $|pred(T_i)| \geqq 2$, T_i is a join task;

If $|succ(T_i)| \geqq 2$, T_i is a fork task;

The Fork-Join structure is one of the basic modeling structures for parallel processing. Most of the structures of the task graph can be simplified to a combination of join and fork structure.

Convention: As a group of ordered tasks can be represented as a DAG, so each node in the DAG corresponds to a task in the order task group, the nodes in DAG are called task or task node ; there are two types of task: process or thread.

The model is based on data flow diagram. Task is executed by a non-preemptive way, where a task node has to wait until all of its predecessor nodes have been executed. Communication overhead between tasks on the same processing node has been assumed zero in this paper.

Figure 2 shows a DAG graph which contains 12 tasks, where each box represents a vertex T_i, and a vertex consists of two parts. T_i indicates the task i, and the number below T_i represents the estimated running time of T_i, which can be expressed as $R(T_i)$. Each arrow represents a directed edge E_i, and parameter attached to the arrow represents $W(T_i, T_j)$ of the directed edge.

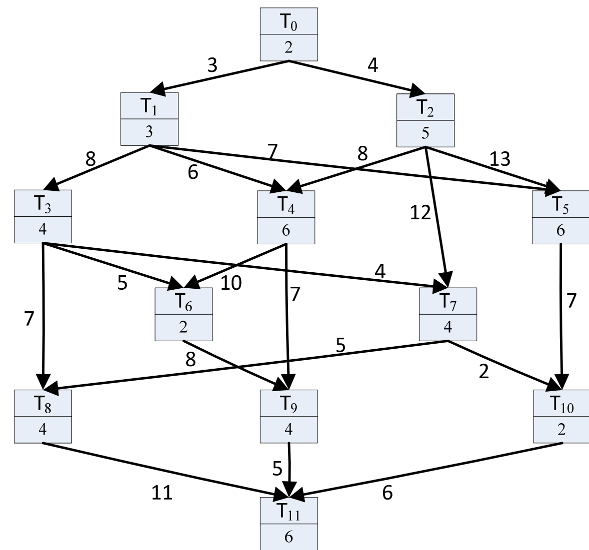

Figure.2 An example of DAG

For generality, we assume there are only one start task and one end task in DAG. If there are multiple start nodes or end nodes, a virtual start node or end node can be added in. It is assumed computing cost of the virtual node is 0; communication edges from start nodes or end nodes to virtual nodes are built, on which communication overhead is 0. Figure 3 shows an example of DAG with two virtual nodes T_0 and T_9 .

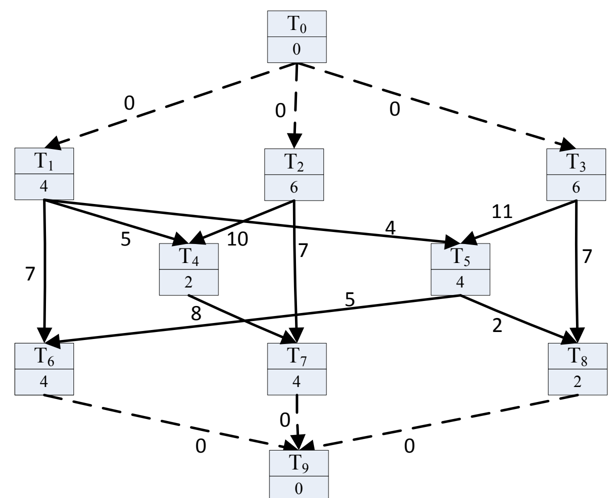

Figure.3 An example of DAG with virtual nodes

The goal function of task scheduling can be expressed as a function f, by which each task node is assigned to an on-duty processing node, and meanwhile both communication overhead and program execution time are minimized. Function f can be expressed as following:

© 2012 ACADEMY PUBLISHER

$f:\{T_i\}\rightarrow\{C_j\}$, i=0,1,2,…,n-1; j=0,1,2,…,k-1; i is the number of tasks, and j is the number of processing nodes. And the following conditions are both met:

1) $\min\sum_{i=0}^{n-1}\sum_{j=0}^{k-1}W_{ij}$

2) min(the overall program running time)

IV. Task Scheduling Algorithm

SL indicates scheduling length of a multi-core-cluster system:

$$SL=\max(SL(P_i)),i=1,2,3,\cdots,m \quad (1)$$

where:

m indicates the number of processor nodes;

$SL(P_i)$ indicates the earliest possible completion time of all tasks which have been assigned to the processor P_i.

$$SL(P_i)=\max(SL(C_{ik})),\ k=1,2,3,\cdots,n \quad (2)$$

where:

n indicates the number of processing cores in P_i;

$SL(C_{ik})$ indicates the earliest possible completion time of all tasks which have been assigned to the processing core k in processor P_i.

$$SL(C_{ik})=\sum_{T_j\in C_{ik}}R(T_j) \quad (3)$$

By analyzing (1), (2) and (3), the theory is derived as follows: Task scheduling length is closely related to the earliest possible end time of tasks, and the earliest possible end time of a task is related to the location and completion time of its parent task, thus, the optimization of the scheduling length is transformed into the optimization of tasks sequence which have been assigned to some processing node in multi-core-cluster systems. Because multi-core-cluster systems contain two-level processing nodes, the algorithm of this paper consists of two optimizations: Process sequence to processor nodes and threads sequence to processing cores. Because task execution time is certain, the algorithm of this paper takes minimizing inter-node communication as the main purpose.

When two tasks are assigned to different processing nodes, the communication delay between them is very large, which on the contrary is slight enough to be ignored if the same node is chosen instead of different ones[18].

The task scheduling algorithm based on task duplication is to eliminate the communication overhead through the above principle. That is, some tasks in DAG are allocated redundantly to some processing nodes, so as to realize the purpose of reducing communication overhead between tasks. This means, task duplication is to copy task when processing nodes are idle, to avoid the transmission of predecessor calculation data, thereby the waiting time of processing nodes is reduced and the task can be started in advance.

The main idea of the proposed algorithm is that: setting the goal of “the current task with the earliest start time”, shorten the whole the task execution time by dividing tasks into different groups.

The proposed algorithm is composed of two steps of operations, in which the processes are assigned to processor nodes in the first step and the threads in processes are assigned to core nodes in the second step respectively. There are two strategy would be used in each step operation, which will be introduced in detail in A and B.

A. *Grouping Strategy*

All the task nodes in the DAG structure graph have been scheduled in turn according to task depth. The task depth is determined by (4).

$$\text{level}(T_i)=\begin{cases}0, & T_i=T_s\\ \max\limits_{T_j\in pred(T_i)}\{level(T_j)\}+1, & \text{others}\end{cases} \quad (4)$$

The task with small sequence number is scheduled preferentially when task depth is the same. By (4), it is known that the task depth of parent tasks must be less than that of subtasks, which ensures the parent task is prior scheduled to the sub-task. The start task is firstly scheduled and the end task is lastly scheduled, which meet precedence relations between tasks in DAG.

And then the following selection policy is adopted to generate scheduling groups or set for each task node.

1) If the node has no predecessor task node, the node itself is regarded as a scheduling group.

2) If the node has only one predecessor task node, then a scheduling group is formed by merging it to the scheduling group of its predecessor node according to their priorities.

3) If the task contains multiple predecessor tasks(this task is join node), scheduling group or set will be generate according to the following operation:

First, select a predecessor task group whose sum of execution time R and communication time W is the largest to form scheduling group with the join node, If there are several predecessor task nodes with the same (R + W), then the predecessor node with bigger W would be selected, which means, that selection can schedule critical task to the same processing node so that the join node starts earlier.

And then select a node from the rest of predecessor task nodes of the join node, which is assumed T_i. The total computing time of the nodes in scheduling group except join node is Q. There are two value are calculated respectively: one is the start time of the join task node when merging the predecessor node into task scheduling group of the join node, expressed as $S1=Q+R(T_i)$, while the other is the start time of the join task node without merging, expressed as $S2=R(T_i)+W$. if $S1\leqq S2$, then merge the predecessor node into task scheduling group of the join node, which runs on the same processing node; if $S1>S2$, then a new scheduling group relevant to the join node should be formed based on the predecessor task, which will be run on other processing nodes. select the rest of the predecessor nodes following the above steps

© 2012 ACADEMY PUBLISHER

until the entire predecessor nodes have been scheduled; finally a join node task scheduling group or a scheduling set is formed.

Redundancy tasks are generated from task duplication. In the process of generating scheduling group, redundant tasks in the same group may be properly eliminated if the start time of successor task nodes will not be delayed, so the final performance of the task scheduling is not affected. Similarly, redundant scheduling groups in scheduling set can also be properly eliminated.

We take Figure 2 as an example to generate scheduling groups or set for each node:

Scheduling group of T_0 is (T_0) because T_0 hasn't predecessor task nodes, and its execution time is 2.

Scheduling group of T1 is (T_0,T_1) because T_0 has only one predecessor task node, and its execution time is 5.

Similarly, Scheduling group of T_2 is (T_0,T_2), and its execution time is 7.

Scheduling group of T_3 is (T_0,T_1,T_3), and its execution time is 9.

Both T_1 and T_2 are predecessor nodes of T_4 node. (R+W) of T_1 is 11 ((R+W) is the execution time of scheduling group of T_1 plus the communication time between T_1 and T_4). (R+W) of T_2 is 15,for 11<15,so obtains the group (T_0,T_2,T_4) by merging T_4 into the scheduling group of T_2. If T_1 is merged into the scheduling group of (T_0,T_2,T_4), and then get (T_0, T_1, T_2, T_4). Now start time S1 of T_4 is 10, that is S1=10, and meanwhile set {(T_0,T_2,T_4), (T_0,T_1)} is obtained without merging, thus S2=11(S2 is start time of T_4);for S1<S2, so that the scheduling group of T_4 is (T_0, T_1, T_2, T_4), and its execution time is16.

Similarly, the scheduling group of T_5 is (T_0, T_1, T_2, T_5), and its execution time is 16. The scheduling set of T_6 is {(T_0,T_1,T_2,T_4,T_6), (T_0,T_1,T_3)}, and its execution time is18. The scheduling set of T_7 is {(T_0,T_2,T_7), (T_0,T_1,T_3)}, and its execution time is 17. The scheduling set of T_8 is {(T_0,T_2,T_7,T_8), (T_0,T_1,T_3)}, and its execution time is 21. The scheduling set of T_9 is {(T_0,T_1,T_2,T_4,T_6,T_9), (T_0,T_1,T_3)}, and its execution time is 22. The scheduling set of T_{10} is {(T_0,T_1,T_2,T_5,T_{10}), (T_0,T_2,T_7), (T_0,T_1,T_3)}, and its execution time is 21. The scheduling set of T_{11} is {(T_0,T_1,T_2,T_5,T_{10},T_{11}) , (T_0,T_2,T_7,T_8), (T_0,T_1,T_3) , (T_0,T_1,T_2,T_4,T_6,T_9)}, and its execution time is 33.

Grouping strategy can be realized by function Group( ), and the concrete realization of Group( ) as follows:

```
void Group (G, scheduling_set)
{
  //input: G = (T, E, R, W)
  //output: a scheduling set
  Ascending order by task depth for all tasks in DAG;
  for(i=0; i<v; i++)
  {
   Construct tasks scheduling groups or set for each task;
   for(j=0; j< in-degree of the current task, j++)
   {
   Seek critical parent task for each current task ;
   Merge the current task to the scheduling group of its
        critical parent task according to their priorities;
   }
  // Eliminate redundant tasks in a scheduling groups
  for(k=0; k<the number of tasks in each scheduling
     groups; k++)
   {
   Count the number of occurrences for each task;
   Eliminate redundant tasks;
   }// end of eliminate redundant tasks
  }// end of construct tasks scheduling groups
}// end of Group()
```

In summary, the process of constructing scheduling groups or set is based on task duplication. A task and its critical parent task are allocated to the same processing node, which minimize the communication overhead, so that current task can be started at the earliest start time, which will shorten the schedule length.

B. Adjusting Strategy

The process, adjusting scheduling set according to the number of processing nodes, can diminish redundant task nodes without influencing the task execution efficiency.

1) When the number of processing nodes is less than that of task scheduling groups in the scheduling set, find out two scheduling groups which contain the most number of same task nodes to merge, so that the execution time of a merged scheduling group is minimized. Because some task nodes have been copied to different scheduling groups, where might appear the same task repeatedly. At last, the number of scheduling groups can be reduced to less than or equal to the number of processing nodes by way of merging.

2) When the number of processing nodes is more than that of task scheduling groups in scheduling set, directly allocate each task group for one processing nodes. The parallel program execution time is the execution time of the task group whose task completion time is the longest.

The threshold is used to control load balancing, which is the number of tasks that each processing node should load, and it fluctuates within a limited range. The threshold values for the processor node and core nodes can be calculated as follows respectively:

$$\text{Threshold}_{\text{processor}} = \max\left(\frac{N_{\text{thread}}}{N_{\text{core}}}, M_{\max}\right) \times (1+\alpha) \quad (0 \le \alpha \le 1) \quad (5)$$

$$\text{Threshold}_{\text{core}} = \frac{\text{Threshold}_{\text{processor}}}{N_{\text{core}}} \quad (6)$$

The M_{max} in expression (5) is the maximum number of threads in a process, and α is a percentage value in expression (5), which is used to balance between load balancing and communication decreasing.

When α is larger, edges with bigger weight are selected by the algorithm, which tends to reduce the communications; When α is smaller, edges with large weight are more likely to be given up, this time the algorithm further aims to balance the load.

© 2012 ACADEMY PUBLISHER

Normally, a solution may be obtained after several times' combine. The solution might become impossible to work out if the load balance conditions are strict, and in that case the threshold should be increased for iterating again.

Adjusting strategy can be realized by function Adjust ( ), and the concrete realization of Adjust ( ) as follows:

```
void Adjust(scheduling set, multiple scheduling group )
{
//input: a task scheduling set
//output: multiple scheduling groups
Count the number of processing nodes;
   if (the number of scheduling groups in the scheduling
       set> the number of processing nodes)
   {
    do
    {
     Find out two scheduling groups which contain the
     most number of same task nodes to merge;
    }while(the number of scheduling groups in the
       scheduling set> the number of processing nodes)
   }// end of if
Assign each task scheduling group to one processing
nodes according to the threshold;
}//end of Adjust ()
```

C. Algorithm Description in General

Multi-core-cluster systems possess the features both shared storage architecture and distributed storage architecture, thus they are supportive for task-level inter-node coarse-grained parallel and intra-node fine-grained parallel. Accordingly, the algorithm is composed of two steps of operations, in which the processes are assigned to processor nodes in the first step and the threads of processes are assigned to processing cores in the second step respectively.

Step1: allocation of processes

1) Construct processes scheduling groups or set on the basis of processes DAG.

2) Adjust scheduling Sets.

3) Assign processes scheduling groups to some processors.

Step2: allocation of threads

1) Change processes that have been allocated to each processor into threads, and generate threads DAG for each processor, so the thread DAG is a part of entirety DAG.

2) Construct thread scheduling groups or set on the basis of threads DAG for each processor.

3) Adjust scheduling sets.

4) Assign thread scheduling groups to some cores.

D Considerations Task Scheduling for Multi-core-SMP-Cluster Systems

The hierarchy of the system becomes more complex in the multi-core-SMP-cluster system, where the processing nodes consist of SMP nodes, processor nodes, and cores nodes. Processors in a SMP node share main memory, and they coupled looser than cores inside of a processor, but tighter than between SMP processing nodes.

Based on the above considerations, the task allocation in multi-core-SMP-cluster systems can be done by a revision of the above the proposed algorithm. At this time, the algorithm is composed of three steps of operations instead of two steps, that is, processes are allocated to SMP nodes in the first step, threads are allocated to processors nodes in the second step, and threads are allocated to processing cores in the third step.

V. Complexity Analysis

Each step of operation both includes function Group( ) and function Adjust( ). In the first step operation, the cycle index of extrinsic cycle in Group( ) is v1 ,and v1 is the number of processes in the process DAG; the cycle index of internal recycle in the worst case is e1, and e1 is the maximum of in-degree in processes DAG, so the time complexity of Group( ) is O(e1*v1). In function Adjust( ), all processes are not related in the worst case, and then the number of scheduling groups is v1. If the number of processor is p1, as above the maximum cycle index of process combination is:

$$\sum_{i=p1}^{v1} i = (v1 + p1) \times (v1 - p1 + 1)/2$$

So the time complexity of adjust( ) is $O(v1^2)$. Thus the time complexity for the first step operation is $O(e1*v1)+O(v1^2)=O(v1^2)$.

Similarly, the time complexity for the second step operation is $O(e2*v2)+O(v2^2)=O(v2^2)$,in which e2 is the maximum of in-degree in threads DAG, v2 is the number of threads in the threads DAG.

The time complexity of the proposed algorithm is $O(v1^2)+ O(v2^2)= O(c^2)$, c is the largest between v1 and v2.

The time complexity of [11][12][15] is $O(v^2)$, where v is the number of all threads in entirety threads DAG, so the time complexity of the proposed algorithm is less than others.

VI. Experiments and Evaluation

Task scheduling has no recognized test project, and the number of actual example sets for test is fairly small, so using random task graph as input set for task scheduling test is a common method[19].

The most basic performance measure of task is the task execution time[20]. Usually, the length of the task scheduling is used to reflect the task execution time, which is expressed as schedule length, and it is also the latest task completion time.

Random graphs are used as input set of task scheduling testing in this experiment, and the algorithm has been evaluated through comparing the scheduling length, namely the latest task completion time[21].

Both the algorithm and genetic algorithm are realized respectively by programming in the experiment. The two algorithms have been evaluated and analyzed by using 1400 task precedence graphs which are randomly generated. The number of threads is between 64 and 1024， and they belong to 16 ~ 64 processes; the weight of edge between two thread nodes or process nodes is

© 2012 ACADEMY PUBLISHER

random in the range of from 0 to 50. The weights of processes node belongs to [20, 30]， The weights of threads node belongs to [10,15]， Tasks have been allocated on a variety of hardware platforms where the number of multi-core processors are from 8 to 32 and the number of processing cores are from 4 to 32， and α=0.15 in (6).

Figure 4 shows the optimal rate and error rate of our algorithm, in which the error rate is calculated as follows according to the sum of weights between threads assigned to different processing nodes (noted as LSW: Left Sum of Weights):

$$errorrate = \frac{\sum(\text{our algorithm_LSW}) - \sum optimal_LSW}{\sum optimal_LSW} \times 100\% \quad (7)$$

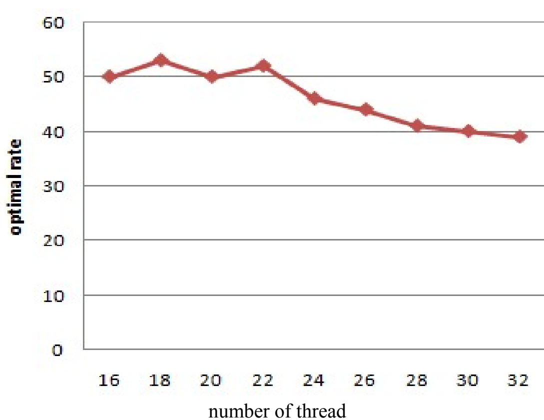

Figure 4. Optimal rate

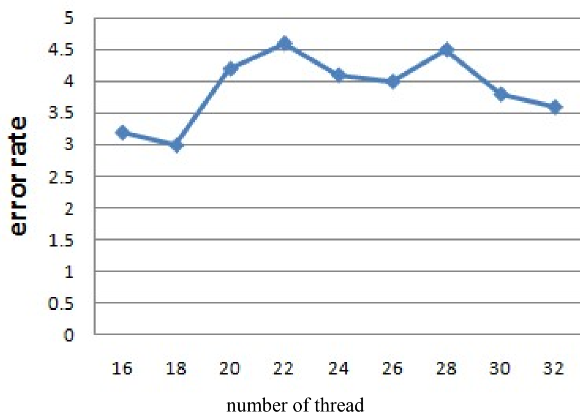

Figure 5. Error rate

As Figure 5 shows, although the percentage of optimal solutions decreases with increasing of thread number, error rates are always lower than 5%. This means that the proposed algorithm can always find near-optimal solutions.

Some of the experiment data are listed in Table 1, where the final task execution time is an average value. Figure 6 shows the static performance curve of the algorithm.

TABLE 1
AVERAGE EXECUTION TIME CONTRAST OF THE PROPOSED ALGORITHM AND GENETIC ALGORITHMS

Number of processor	Number of core	Number of process	Number of thread	SL of This Algorithm /ms	SL of genetic algorithm /ms
8	4	16	64	48	72
	8	24	185	56	85
16	12	32	310	69	90
	16	40	400	93	110
24	20	48	512	123	145
	24	52	726	189	261
32	28	56	896	232	312
	32	64	1024	278	362

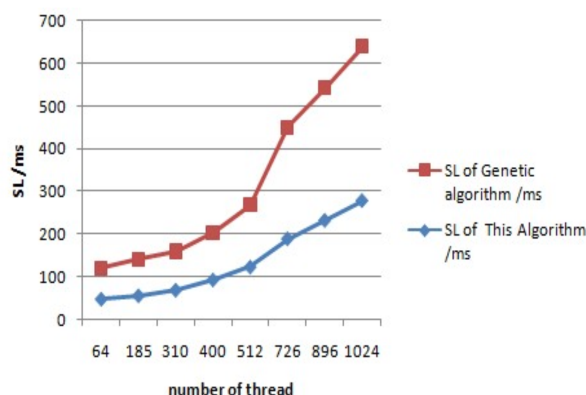

Figure 6. The static performance curve of the algorithm

We can see from table 1 and figure 6 that the algorithm can find better solution than genetic algorithm, that is, the algorithm could get shorter final task execution time. From figure 6 we can find that when the task is fewer, the algorithm performance is more similar to genetic algorithm, but with the increase of the number of tasks, this algorithm is obviously better than genetic algorithm. So this algorithm has better performance in the CMP with large number of tasks.

In the experiment, when the weight of edge between two threads or processes is increased from [0, 50] to [60,100], the rate of SL of the proposed algorithm and SL of genetic algorithm is smaller than before, which demonstrates that the performance of the proposed algorithm is better as the ratio of total communication cost and total computation cost in the task schedule model becomes larger.

VII. CONCLUSION

This paper discusses the task scheduling problem in multi-core-cluster systems, builds the task scheduling model, and then proposes a task scheduling algorithm based on task duplication which consists of two steps of operations, and respectively establishes a mapping from task scheduling groups to some processing nodes. For the algorithm, minimization scheduling length is the primary objective, and keeping load balancing between processing nodes is secondary objectives. The time complexity of this algorithm is less than similar algorithms.

© 2012 ACADEMY PUBLISHER

Comparison test shows that the algorithm can obtain near-optimal solutions in reasonable time, and behave even better in scalability than genetic algorithms. Furthermore, while the ratio of total communication cost and total computation cost in the task schedule model becomes larger, the advantage of this algorithm is more obvious.

REFERENCES

[1] S. Y. Borkar, P. Dubey, K. C. Kahn, D. J. Kuck, H. Mulder,S. S. Pawlowski, and J. Rattner. Platform 2015: Intel®processor and platform evolution for the next decade, White Paper, Intel Corporation, 2005.

[2] Dummler, J., Rauber, T., Runger, G..Mapping Algorithms for Multiprocessor Tasks on Multi-core Clusters. In:Parallel Processing, 2008. ICPP '08. 37th International Conference

[3] T. Agarwal, A. Sharma, and L. V. Kale. Topology-aware task mapping for reducing communication contention on large parallel machines. In Proc. of the 20th Intl. Parallel and Distributed Processing Symposium (IPDPS 2006). IEEE, 2006.

[4] H. Chen, W. Chen, J. Huang, B. Robert, and H. Kuhn. Mpipp: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters. In ICS '06: Proc. of the 20th Int. Conf. on Supercomputing, pages 353-360, New York, NY, USA, 2006. ACM.

[5] N. Vydyanathan, S. Krishnamoorthy, G. Sabin, U. Catalyurek, T. Kurc, P. Sadayappan, and J. Saltz. Locality conscious processor allocation and scheduling for mixed parallel applications. In Proc. of the 2006 IEEE Int. Conf. on Cluster Computing. IEEE, 2006.

[6] Gerasoulis A, Yang T. A Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessors[J].Journal or Parallel and Distributed Computing,Dec.1992,16:276-291

[7] YUAN Yun, SHAO Shi, Tasks scheduling algorithm for parallel system with multi-core processor, ComputerApplications, 2008,28(12),280-283

[8] Pasham,S.,WM.Lin,"Efficient task scheduling with duplication for bounded number of processors",Proceedings.11th International Conference on Volume 1, 20-22 July 2005 Page(s):543-549 Vol.1

[9] Darbha S, Agrawal D P. Optimal scheduling algorithm for distributed-memory machines. IEEE Transactions on Parallel and Distributed Systems, 1998, 9(1): 87-95

[10] Park C-I, Choe T-Y. An optimal scheduling algorithm based on task duplication. IEEE Transactions on Computers, 2002, 51(4): 444-448

[11] Zhou Shuang-E, Yuan You-Guang, Xiong Bing-Zhou, Ou Zhong-Hong. An algorithm of processor pre-allocation based on task duplication. Chinese Journal of Computers, 2004, 27(2): 216-223

[12] Ahmad I, Kwork Y K. On exploit task duplication in parallel program scheduling. IEEE Transactions on Parallel and Dis-tributed Systems, 1998, 9(9): 872-892

[13] Kruatrachce B, Lewis T. Grain size determination for paral-lel processing. IEEE Software, 1998, 1: 23-32

[14] Li Min,Wang Hui,Li Ping. Tasks mapping in multi-core based system:hybrid ACO&GA approach[C].In Proceedings of 5th International Conference on ASIC, Oct 2003.

[15] Y. Liu, X. Zhang, H. Li, and D. Qian. Allocating Tasks in Multi-core Processor based Parallel Systems. In 2007 IFIP International Conference on Network and Parallel Computing Workshops (NPC2007), 2007.

[16] Zhao Lei, Research on task scheduling for multi-core processor, Harbin university of science and technology [D].,2010.

[17] Ennals Robert, Sharp, Richard, et al. Task partitioning for multi-core network processors[C].In Proceedings of 14th International Conference on Compiler Construction, April 2005.

[18] Attiya G, Hamam Y. Two phase algorithm for load balancing in heterogeneous distributed systems//Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing. A Coruna, Spain, 2004: 434-439

[19] Tsuchiya, T., Osada, T., Kikuno, T. A new heuristic algorithm based on GAs for multiprocessor scheduling with task duplication. This paper appears in: Algorithms and Architectures for Parallel Processing, 1997. ICAPP 97., 1997 3rd International Conference on

[20] Bansal S, Kumar P, Singh K. An improved two-step algorithm for task and data parallel scheduling in distributed memory machines [J]. Parallel Computing, 2006, 32(10):759-774

[21] LAN Zhou, SUN Shi-Xin. An Algorithm of Allocating Tasks to Multiprocessors Based on Dynamic Critical Task, journal of computers,2007,30(3):454-461

© 2012 ACADEMY PUBLISHER